SEARCHING GENERAL CELLULAR AUTOMATA RULES USING EVOLUTIONARY COMPUTATION

Berlanga A., Isasi P., Sanchis A., Molina J.M.

***Abstract*-- In this work a general framework, called Uniform Coevolution, is introduced to overcome the testing problem. This framework is based on competitive evolution ideas where the solution and example sets are evolving by means of a competition to generate difficult test beds for the solutions in a gradual way. The method has been tested with the density parity problem in cellular automata, where the selected examples can biased the solutions founded. The results show a high value of generality using Uniform coevolution, compared with no coevolutive approaches.**

***Index terms*—competitive coevolution, cellular automata, density classification problem**

I. INTRODUCTION

Coevolution refers to the simultaneous evolution of two or more species, where the survival of each species depends one each other. This process has proven its capabilities of generating complexity in nature. From the computational point of view, a competitive system is composed of several confronted subsystems where the evaluation of a subsystem depends on the performance over the opposite one. The ideas of coevolution have been introduced into the field of evolutionary computation from different points of view:

Applying coevolution: One of the first authors in applying coevolution in an optimization problem was Hillis in his work about coevolution of parasites for improving solutions in a sorting networks problem [1]. In this work, the competition takes place between individuals solutions and individuals examples. These examples were generated to be a good test bed for the solutions. In this way both, populations of solutions and population of examples were competing to beat one each other. The result was a better method to generate solutions for the sorting nets problem. Further works have applied these ideas to other optimization problem [2].

Coevolution in game theory: Some related works have been done in the field of game theory. This field is a good example of domain in which the ideas of coevolution could be easily tested. Angeline [3], used coevolutive ideas in the game of Tic Tat Toe. In this work, individuals are different game strategies, they compete one against each other. Therefore, the fitness of each individual does not have an absolute value, but a value depending on the goodness of their opponents. These ideas have been extended in other games like the game of Othello [4] or the game of Tag [5] and even generating theories of learning in games, in competitive environment [6].

Theoretical works: More recently, some works for establishing the theoretical foundations on coevolution have been done. Paredis, in 1996, proposes a general framework for the use of coevolution to boost the performance of genetic search [7], and introducing a new type of Genetic Algorithm called Co-evolutionary Genetic Algorithm. Rosin and Bellew [8] introduce three new techniques in competitive coevolution: Competitive fitness sharing, shared sampling, and hall of fame; and provide several different motivations for these methods and mathematical insights into their use.

In the present article, a new method, based on Hillis's ideas to use coevolution for overcoming the testing problem is introduced. We propose a general framework to apply coevolution to any evolutionary computation technique, in such a way that the examples could evolve towards optimal data examples.

II. THE TESTING PROBLEM

In some previous works [9], [10], [11], we have studied evolutionary systems and founded the over-adaptation problem. In these cases, some rules (or neural networks) have been evolved for navigation problem in robotics (or learning rules for neural networks have been found). In all cases, the achieved solutions were ad hoc for specific situations. For instance, in the robot navigation problem, when the robot had to face new situations, new rules had to be evolved. All these problems are referred in the literature as the testing problem.

To overcome the testing problem Rosing and Bellew [8] suggest a new Co-evolutionary method, the shared sampling. In this method a population of examples is always kept. Each example of the population is evaluated computing its performance over a previously selected set of solutions. In the same way, each solution is evaluated computing its performance over a previously selected set

Computer Science Dept. Universidad Carlos III de Madrid
Avda. Universidad, 30, 28911, Leganés, Madrid, Spain
e-mail: aberlan@ia.uc3m.es

of examples. The selection of the examples is carried out proportionally to the evaluation of examples. The examples with better general evaluations are preferred as test cases for the solutions, and their evaluations are computed again.

The previously mentioned Hillis' solution [1] is similar. In this case, the examples are not selected, each time an evaluation of a solution is needed. By the opposite, each solution has a subset (subpopulation) of examples related with it. This subset is kept constant, and it is in continuous evolution.

The evolution of the examples, in both cases, tries always to generate harder examples for the solutions. As the solutions are more complex and accurate, they must prove their capabilities with more sophisticated and complex examples.

In some cases the above-proposed solutions could generate a serious problem. For instance in a problem where the generation of good solutions over reduced examples set is a very difficult task. In this case:

- If the examples evolve towards hard data sets, the process could end into an impossibility of achieving solutions for these hard example sets, and the continuous adaptation of the examples could stop the adaptation of the solutions. These are cases in which the fitness landscape is abruptly sharpened by the examples.

- If the adaptation of examples process is carried out in such a way that the adaptation of solutions is allowed the solutions could reach in a process of over-adaptation, making more difficult the generation of more accurate solutions. In these cases, the examples are not modified too much in order to allow the generation of solutions, but the generated solutions become good for the particular example set, and solutions are not able to solve the problem for different examples.

These two problems imply some difficulties in the process of evaluation of the solutions:

- Firstly, as the examples set is shared for all solutions, even if the evolution of examples was slowed down, solutions could be faced with too many difficult examples. A good example for a solution could be a bad one for another. When the examples are too difficult, there is no selective pressure for solutions to evolve. The fitness values of all solutions are similar, low fitness, and there is no way of selecting good individuals to improve solutions.

- Secondly, when a population of solutions has been adapted to an examples set, the change of the example set could have negative consequences. The fitness landscape changes abruptly and the previously evolved solutions have no meaning of evolving toward the new fitness landscape.

For overcoming these problems, we propose, in this work, a new method of adjusting coevolution to allow both the evolutions of good solutions and hard test examples in difficult generalization problems.

III. UNIFORM COEVOLUTION

The architecture is composed of a population of solutions and a set of populations of examples (one population of examples for each individual in the population of solutions). This structure reflects what it was called Independent Examples Sets.

The solutions and examples systems are named respectively:

- Solutions Generator System (SGS). - It is composed by a population of solution individuals (SI). For computing their fitness, is necessary to face each individual with a set of different situations, examples, represented by a population in the Examples Generator System.

- Examples Generator System (EGS). - It is a meta-population composed of meta-individuals, which are populations of examples (PE). Each PE is related with a SI. Example individuals (EI) compose the PE. The fitness of those individuals is inversely proportional to their related SI fitness, when operating over them.

The evolution of each system depends on the other's evolution. The general procedure is as follows:

1. Initialization of the populations:
   1.a. SGS initialization (m SI individuals)
   1.b. EGS initialization (m PE of n EI individuals each)
2. Computation of the fitness
   2.a. Evaluation of the SI over each individual EI in its related PE
   2.b. The fitness of the SI is a combination of the above evaluations
   2.c. The Fitness of the PE is set inversely to the fitness value of the correspondent SI
3. Generation of new populations
   3.a. PE evolutions by means of generation of new EI's applying an ad-hoc genetic operator (Incremental Genetic Operator -IGO-)
   3.b. EGS and SGS evolution

3.1 Solution Generator System

The objective of this system is to gradually generate better solutions to a particular problem. Any evolutionary computation method can be used, where an individual represents one problem solution. The evolution of the SGS follows the dynamics of the evolutionary computation method selected.

Computation of solution's fitness: The generation of better solutions is driven by the evaluation function, also called fitness function. Each individual is evaluated over a set of examples. Lets call PE_i the examples set of the individual *i*, this population is composed of several independent blocks (*A..Z*), which meaning will be explained later. Therefore EI_{ij}^{A} is the j-th example of the block *A* for the set PE_i. As previously mentioned, for the *smoothing fitness landscape* mechanism, a linear combination of evaluations is used as fitness value of the individual. The fitness for an individual *i* is computed using the evaluation values of that individual over a set of *n* examples, in equation 1.

$$F(SI_i) = \sum_{j=1}^{nb} \frac{F_B^j(SI_i)}{nb} - C\sigma_B^i \quad (1)$$

Where $F(SI_i)$ is the final fitness of the *i-th* solution individual, *nb* is the number of blocks in the population of examples, $\sigma^B{}_i$ is the deviation of the fitness values of the blocks for SI_i, *C* is a constant measuring the importance of the deviations over the normalized total fitness of the blocks, and is $F^j{}_B(SI_i)$ the fitness of SI_i for the block *j*. This value is computed following equations 2 and 3.

$$F_B^j(SI_i) = \sum_{k=1}^{nex} \alpha_{ik}^j f(SI_i, EI_{ik}^j) \quad (2)$$

$$\alpha_{ik}^j = \frac{w_{ik}^j}{\sum_{k=1}^{nex} w_{ik}^j} \quad (3)$$

Where $f(SI_i, EI^j{}_{ik})$ is the value of the evaluation of SI_i over the example $EI^j{}_{ik}$, *nex* is the number of examples of each block. The $w^j{}_{ik}$ values are used to weight the importance of each example in the total computation of the fitness of a *SI*. The *w* values depend on the proximity between the fitness they are weighting and the maximum fitness, and they are computed by the equation 4.

$$w_{ik}^j = \frac{\left(e^{a_i} - 1\right)\left(e^{\beta_j^i} - 1\right) + \left(e^{1-a_i} - 1\right)\left(e^{1-\beta_j^i} - 1\right)}{(e-1)^2} \quad (4)$$

Where a_i is a measure of the evolution degree of the individual over its examples set, and the β_j^i gives an idea about how the example *j* contributes to the total fitness of individual *i*. The a_i values are computed by equation 5.

$$a_i = 1 - \frac{\sum_{k=1}^{n} exf(SI_i, EI_{ik}^j)}{nex \cdot F_{max}} \quad (5)$$

And the β values by equation 6.

$$\beta_j^i = \begin{cases} 0; & \text{if } f_{i,max} = f_{i,min} \\ \dfrac{f(SI_i, EI_{ik}^j) - f_{i,min}}{f_{i,max} - f_{i,min}}; & \text{if } f_{i,max} \neq f_{i,min} \end{cases} \quad (6)$$

Where F_{max} is the maximum fitness value that a SI_i could ever reach, $f_{i,max}$, $f_{i,min}$ are the maximum and minimum fitness values reached for SI_i over it related examples set respectively, described through equations 7 and 8.

$$f_{i,max} = Max\left\{f(SI_i, EI_{ij})\right\}_{j=1}^{N} \quad (7)$$

$$f_{i,min} = Min\left\{f(SI_i, EI_{ij})\right\}_{j=1}^{N} \quad (8)$$

3.2 ExampleS Generator System

The second subsystem in Uniform Coevolution is called Examples Generator System (EGS). The EGS is a meta-population composed of a set of populations $\{PE_i\}$. Therefore, the EGS is composed of two dependent evolutive systems: the meta-population and the PE_i, one embedded into the other.

Meta-population. As a way of developing the *independent examples sets* idea, the examples are divided in M independent sets PE_i, which are the individuals in the meta-population. Each PE_i is related and competes against a unique solution SI_i. Individuals PE_i is composed of a number of chromosomes. These chromosomes are the previously mentioned blocks of the PE_i. Each chromosome represents a set of examples. These blocks, that could be considered independent and evolves in an independent way, are needed for crossover purposes. When the individuals in the meta-population interchange their genetic material, the blocks are interchanged.

Population PE_i. All the individuals EI_{ij} in a block are generated from an especial individual called "seed example". The generation of EI_{ij} is based on a particularly designed Genetic Operator called Incremental Genetic Operator IGO. This generation process constitutes the unique method for evolving PE_i.

Evolution of population. Initially all the seed examples of the blocks are identical and randomly generated. The individuals in the blocks are all the same and equal, in this initial step, to their related seed example.

In furthers steps of evolution, the individuals in a block are generated from the seed example by the Incremental Genetic Operator (IGO). Also the blocks of PE_i are inherited by the offspring from their parents.

Incremental Genetic Operator. For the designing of the IGO is necessary to define a distance function between examples. This distance is a measure of the differences existing among examples: most different are two examples a higher value outputs the function and vice-versa, equation 9.

$$E \times E \rightarrow \Re \quad (9)$$

Where *E* is the set of all possible examples for a particular problem.

As the distance between examples is a numerical value, the change in the examples could be computed using the equation 10.

$$I = \frac{B^2\left(e^{\frac{2 f_b}{F_{max}}\ln\left(\frac{A-B}{B}\right)}-1\right)}{A-2B} \qquad (10)$$

Where A and B are two constants to regulate the shape of the function. This shape conforms how different the examples have to be inside a block from the fitness of the individual. The value calculate is used to generate examples which a distance between genotipes equal to I.

For the evolution of examples the following rules are used:

1. To generate the first example for the individual I (SI_i)
2. To generate all the individuals in the block, the equation 11 is used.

$$N(0,I_j) = D(E_i,E_{i+1}) \qquad (11)$$

Where $N(0,I)$ is a Normal distribution, means 0, and deviation I, and $D(x,y)$ is the distance between examples x and y. In other words, the examples are generated in such a way that their distances follow a Normal distribution of deviation I_j computed for SI_j.

IV. DENSITY CLASSIFICATION PROBLEM

The Density Classification Problem (DCP) is one of the most studied problems in Cellular Automata (CA) [12]. This problem is interesting from both, theoretical and practical aspects, and it has been proven the non-existence of any rule able to solve the problem for a binary CA with a neighborhood of radius one [13]. The DCP is defined by the equation 12.

$$T_{\rho_c}(N,M) = \begin{cases} \Psi_m(s_0) = 0^N & \text{if } \rho(s_0) < \rho_c \\ \Psi_m(s_0) = 1^N & \text{if } \rho(s_0) > \rho_c \\ \text{not determined} & \text{if } \rho(s_0) = \rho_c \end{cases} \qquad (12)$$

Where $T_{\rho c}(N,M)$ is an unidimensional DCP of size N, with a critical density of ρ_c and after M updating periods. If the initial density $\rho(s_0)$, is smaller than the critical density, the CA has to transit, after M steps, to a configuration of all zeros. s_0 is the initial configuration, the configuration of a CA after some i steps is $s_i=\Psi(s_0)$, where the function Ψ defines the rule of the CA.

Figure 1 shows the application of Uniform Coevolution to DCP. A canonical genetic algorithm is applied in SGS, EGS evolve with mechanisms previously described in general Uniform Coevolution architecture.

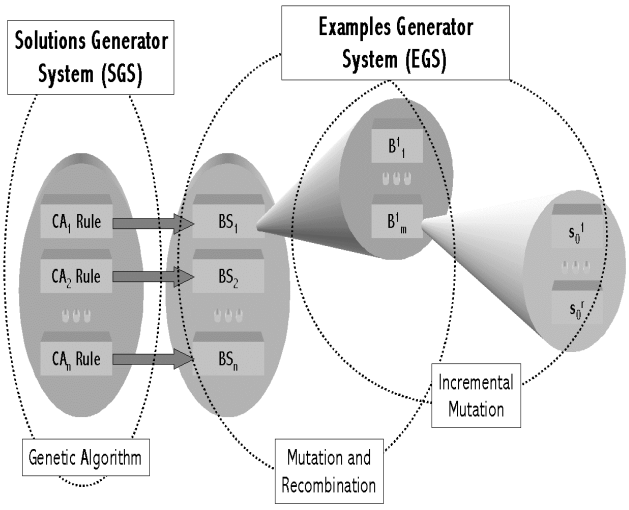

Figure 1: Uniform Coevolution architecture applied to DCP.

A. Experimental environments

Six kinds of experiment have been performed. In table 1 is shown a brief description of each type of experiment.

Table 1: Experiments description.

Experiment	Examples set	Dom. Know.
FixedNI	Constant	No
Fixed	Constant	Yes
RamdomNI	Random	No
Random	Random	Yes
CoevNoSPC	Evolutive	No
CoevUni	Evolutive	No

A total of 30 experiments of each type have been realized to overcome the stochasticity of the GA. In all the experiments a GA have been used to evolve the solutions. In tables 2 and 3 the GA and CA parameters of runs are shown.

To test the generalization capability of the obtained CA rules a total of 1000 new initial configurations are generated.

Table 2: GA parameters.

Population size	100
Chromosome length	128
Crossover probability	0.9
Mutation probability	0.01
Elitism	0.05
Generations	500
Selection operator	Tournament (3)

Table 3: CA parameters.

States	Binary
Neighborhood	7
Dimension	Uni-dimensional
Number of rules	3.4×10^{38}
Boundary conditions	Periodic
Lattice size	149
Number of initial configurations	7.1×10^{44}

These initial configurations are the same for all the experiments and are equally distributed in ten density intervals. This supposes one hundred configurations of density between 0.0 and 0.1, one hundred between 0.1 and 0.2 and so on. The percentage of successfully classified configuration is the measure of generalization of the rule.

Table 4: Generalization results

Experiment	Best Value	Average Value	Sigma
FixedNI	93.46%	91.68%	0.92
Fixed	93.98%	70.74%	20.43
RamdomNI	93.01%	92.10%	0.27
Random	94.10%	92.55%	0.51
CoevNoSPC	93.48%	92.55%	0.42
CoevUni	93.61%	92.63%	0.51

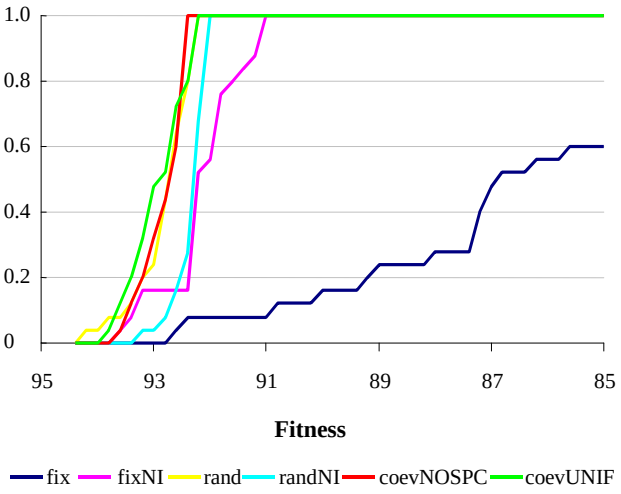

Figure 2: Accumulative probability of finding a specific solution.

The results of this generalization test are summarized in table 4. There are not fundamental differences in best values for all the experiments, however from the average values it can be inferred some conclusions. For the *FixedNI* experiment the average value is shorter. In this case some times the solutions correspond with a local minimum in generalization. This is due to a problem of over-fitting. With the elimination of information in the fitness function, the learning process become less accurate, and in generalization the over-fitting problem can be overcome. Besides, the Co-evolutive experiments show a better generalization capability, and the inclusion of the SPC mechanisms helps to achieve this higher generalization level. These best results are achieved even without including any domain dependent information in the fitness function for the Co-evolutive experiments. It is expected that with such that information the results could be improved.

In stochastic methods the information of the best solution found can be non-realistic. To really compare the efficiency of some methods is better to have an estimation of the easiness of finding a solution in one run. In figure 2, the probability of each solution is shown. For the Co-evolutive experiments all the solutions have a value of generality higher than 92.5%, and there is a probability of 0.1 of finding solutions of above 94%.

V. CONCLUSIONS

The Uniform Coevolution method allows the production of generalized behaviors gradually, changing by time the selective pressure of the individuals. At the beginning, good behaviors are searched, without taking into account the environment. The selective pressure is addressed to the generation of good behaviors in simple environments. As the individuals are adapting to environments, the selective pressure change to search good individuals in gradually more different environments.

In this way, although changing, a similar and smooth fitness landscape is always kept, that allows the adaptation at the same time at the generation of more complex and generalized behaviors. This solves the problem of over-adaptation. The results show how the learning process in Uniform Coevolution goes gradually and uniformly achieving generalization and avoiding local minimal.

The method can be used in any inductive learning problem that uses evolutionary computation mechanisms, to solve the testing problem. The advantage arises from the gradually and independent evolution of the examples, adapting to each particular solution tested. The results have been presented from the solutions point of view, but the analysis of the examples sets finally generated could give some important information about the problem being studied.

VI. BIBLIOGRAPHY

[1] W. D. Hillis, “Co-evolving parasites improve simulated evolution as an optimization procedure”, Artificial Life II, ed. C.G. Langton, Addison-Wesley, Santa Fe Institute, Reading, MA, pp 313-324, 1991.

[2] M. A. Potter and K. A. De Jong, “A cooperative coevolutionary approach to function optimization”, Proceedings of the third conference on parallel problem solving from nature, Y. Davidor and H.P. Schwefel and R. Manner, vol 866, Lecture notes in computer science, Springer Berlag, Berlin, pp. 249-257, 1994.

[3] P.J. Angeline and J.B. Pollack, “Competitive environments evolve better solutions for complex tasks”, Proceedings of the Fifth International Conference on Genetics Algorithms, ed. S. Forrest, Morgan Kaufmann, San Mateo, CA, pp: 264-270, 1993.

[4] R. E. Smith and B. Gray, “Co-adaptive genetics algorithms: An example in Othello strategy”, Proceedings of the Florida Artificial Intelligence Research Symposium, 1994.

[5] C. W. Reynolds, “Competition, coevolution and the game of Tag” Artificial Life IV, ed. C.G. Langton, Addison-Wesley, Santa Fe Institute, Reading, MA, 1996,

[6] Rosin C.D., Belew R.K. “A competitive approach to game learning”. Proceedings of the 9th Annual ACM Workshop on Computational Learning Theory. New York: Association for Computing Machinery, 1996.

[7] J. Paredis, “Coevolutionary computation”, Artificial Life, vol. 2 , pp: 355-375, 1996.

[8] Rosin C.D., Belew R.K. “New Methods for Competitive Coevolution”. Evolutionary Computation (pp. 1-29) 5(1), 1997.

[9] Isasi P., Berlanga A., Molina J.M. And Sanchis A. “Robot Controller against Environment, a Competitive Evolution”. Special Session on Evolutionary Computation, 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics, Alemania , pp 349-354,1997.

[10] Molina J.M., Sanchis A., Berlanga A., Isasi P. “Evolving Connection Weight Between Sensors and Actuators in Robots”. IEEE International Symposium on Industrial Electronics. 1997.

[11] Berlanga A., Molina J.M., Sanchis A., Isasi P. “Applying Evolution Strategies to a Neural Network Robot Controller”. 5th International Work-Conference on Artificial and Neural Networks, IWANN’99. Alicante, España. 1999.

[12] J. Paredis, “Coevolving cellular automata: Be aware of the red queen!”, Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kaufmann, pp: 393-400, 1997.

[13] Land M., Belew R.K. “No Perfect Two-State Cellular Automata for Density Classification Exists”. Physical Review Letters. 74:25. (pp. 5148-5150, 1995.